《大数据分析方法 A（双语）》
实验报告

组员 1 姓名　张沛　学号　20221111057

组员 2 姓名　施炫彤　学号　20221111077

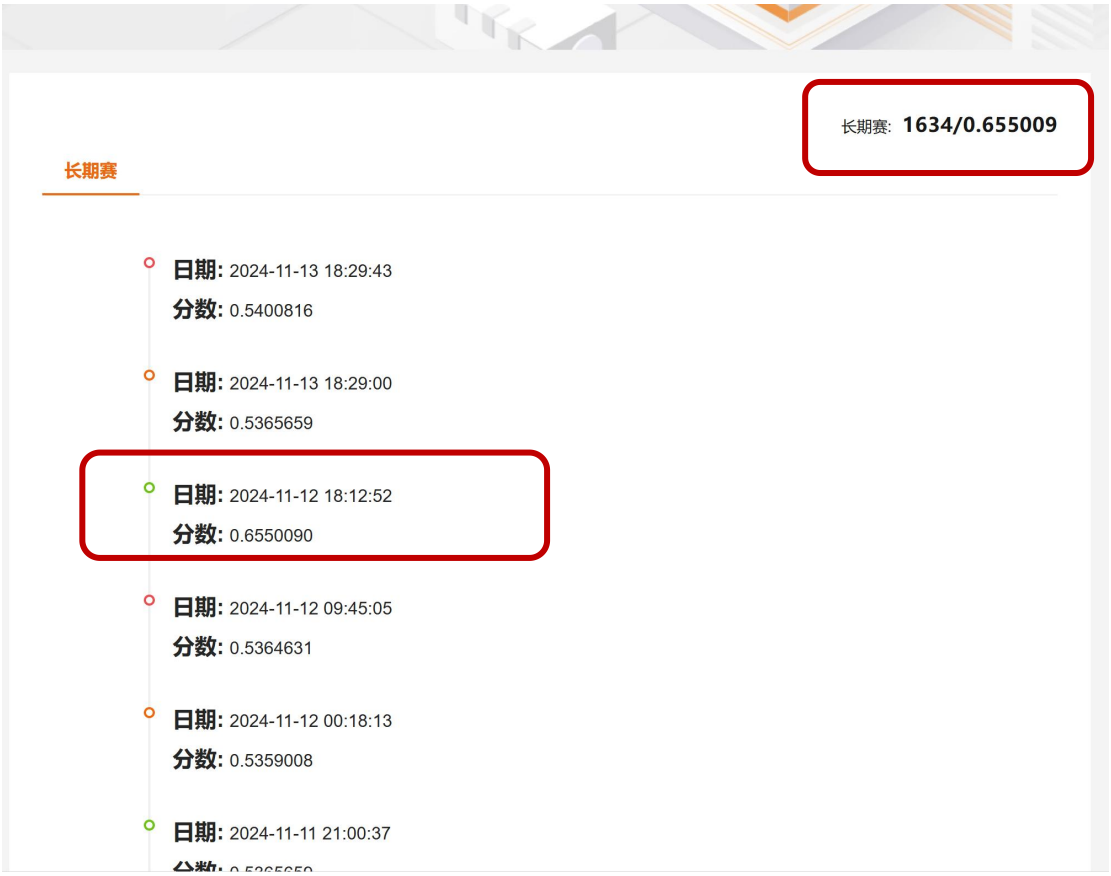
组员 3 姓名　王欣然　学号　20221111116

组员 4 姓名　曹昊天　学号　20211081090

姓名\评分	组员 1 给分	组员 2 给分	组员 3 给分	组员 4 给分	平均分
张沛	\	100	100	100	100
施炫彤	100	\	100	100	100
王欣然	100	100	\	100	100
曹昊天	100	100	100	\	100

网站分数：　0.655009

评分截图：



组员姓名	使用模型	网站得分
曹昊天	随机森林+XGboost	0.655009
张沛	LightGBM	0.648890
王欣然	XGboost	0.631032
施炫彤	神经网络	0.540081

1 问题描述

在当今的电子商务平台上，商家们往往会利用双十一、黑色星期五等特定的促销活动来吸引大量的消费者。这些活动通常会提供各种优惠和折扣，以期在短时间内吸引大量顾客的关注和购买。然而，尽管这些活动能够带来大量的订单和销售额，但许多消费者在这些活动中的购买行为往往是一次性的。也就是说，他们可能只在这些特定的促销期间购买一次商品，之后便不再继续购买。

在本次实验课程中，我们面临的核心问题是如何利用六个月的数据来预测用户在未来一段时间内是否会再次在天猫平台上购买商品。这一任务对于电商平台来说至关重要，因为它能够帮助平台更深入地理解用户的行为模式，从而优化营销策略，提升用户留存率和增加销售额。通过准确预测用户的复购行为，平台能够更好地满足用户需求，制定更有效的营销计划，从而提高用户的忠诚度和平台的整体收益。

2 问题分析与解决思路

2.1　问题分析

本问题属于典型的二分类问题，其核心目标在于区分用户是否会产生复购行为，同时还要给出相应的预测概率。

可以用哪些模型？

在解决二分类问题时，有许多机器学习模型可供选择。以下是一些常用的模型：

•逻辑回归：这是一种广泛使用的线性分类模型，适用于二分类问题。它通过使用逻辑函数来预测一个事件发生的概率。

•随机森林：这是一种集成学习方法，通过构建多个决策树并进行投票来提高预测准确性。随机森林在处理高维数据和非线性关系方面表现出色。

•梯度提升树：这是一种强大的集成学习方法，通过逐步添加弱学习器来构建强学习器。梯度提升树在许多分类和回归任务中表现出色，尤其是在处理复杂数据结构时。

在选择模型时，需要考虑数据的特性、模型的复杂度以及预测性能。例如，如果数据集较小，逻辑回归可能是首选，因为它简单且易于解释。对于更复杂的数据结构，随机森林或梯度提升树可能更为合适。而当数据集非常大且需要处理非线性关系时，神经网络可能提供更好的预测结果。为了确定最佳模型，我们通常会进行交叉验证，并使用诸如准确率、召回率、F1 分数等指标来评估模型性能。

2.2　解决思路：

为了解决这个二分类问题，我们解决思路主要分为以下几个部分：

（一）特征工程：

在特征工程阶段，我们需要提取和构造与复购行为相关的特征。这些特征可能包括用户行为特征，如浏览历史、购买频率和点击率；商品特征，如价格、品牌和类别；用户画像特征，如年龄、性别和地理位置等。

（二）模型选择：

在选择合适的机器学习模型进行训练和预测时，我们需要考虑数据的特性和问题的复杂性。常用的模型包括逻辑回归、随机森林、梯度提升树等。我们小组的四位成员分别使用了Xgboost、随机森林+XGBoost、LightGBM、神经网络等多种方法来进行分析；

（三）模型训练与验证：在模型训练阶段，我们使用训练数据集对选定的模型进行训练。为了评估模型的性能，我们还需要在验证集上进行测试，以确保模型具有良好的泛化能力。

（四）算法改进：根据模型评估结果，我们可能需要调整模型参数或尝试不同的算法以提高预测准确性。这可能包括调整学习率、树的深度、特征选择方法等。

（五）结果分析：在模型训练和验证完成后，我们需要对模型预测结果进行深入分析，以提取有价值的商业洞察。这可能包括识别影响复购行为的关键因素，以及预测哪些用户更有可能进行复购。

3　数据探索

3.1　数据读取与缺失值分析

从四个文件读取数据后，对缺失值情况进行了检查，输出结果如下：

```
User log missing values:
 user_id           0
item_id            0
cat_id             0
merchant_id        0
brand_id        1327
time_stamp         0
action_type        0
User info missing values:
 user_id          0
age_range     2217
gender        6436
Train missing values:
 user_id         0
merchant_id     0
label           0
Test missing values:
 user_id             0
merchant_id         0
prob           261477
```

① 用户行为日志的 brand_id 字段有 1327 个缺失值。

② 用户画像中的 age_range 缺失 2217 个，gender 缺失 6436 个。

③ 训练集无缺失值；测试集的 prob 字段为空，因为这是需要预测的结果。

为确保数据完整性，缺失值处理如下：

④ gender 缺失值填充为 2，表示未知性别。

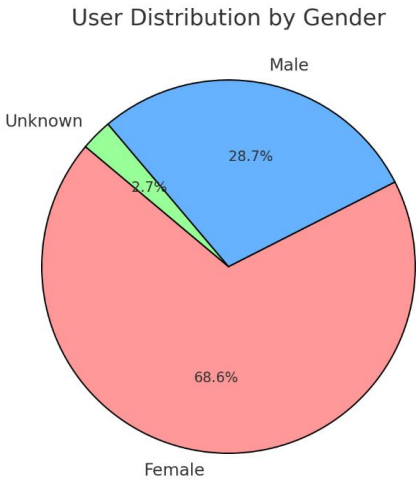
⑤ age_range 缺失值填充为 -1，表示未知年龄。

3.2　数据分布与可视化分析

为深入理解用户和商家的数据分布，利用可视化图表分析各个关键特征。以下是数据探索中生成的几张关键图表及其解读。

· 性别分布

性别分布的饼图揭示了不同性别用户的比例：



女性：68.6%（粉红色）

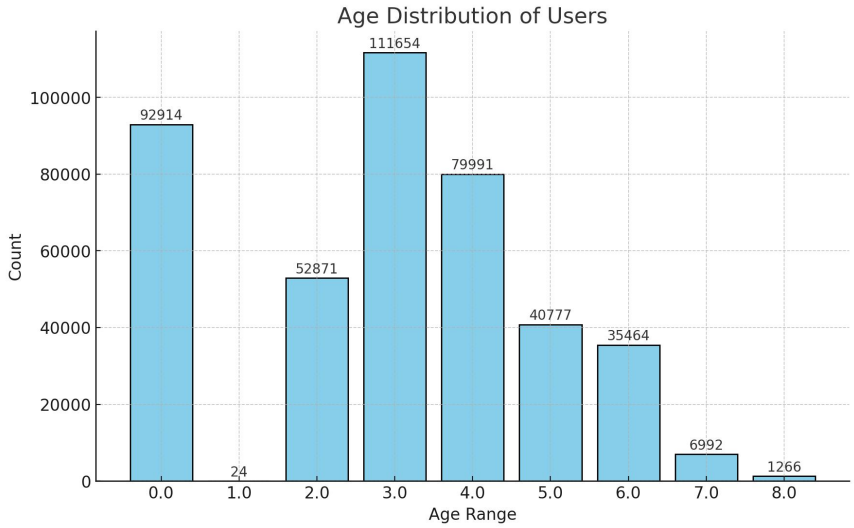
男性：28.7%（浅蓝色）

未知性别：2.7%（浅绿色）

分析：图中显示，女性用户占比接近 70%。这提示我们可以在特征工程中关注性别差异，设计针对女性用户的特征，分析其行为特点。

· 年龄分布

年龄分布的柱状图展示了不同年龄段用户的数量：

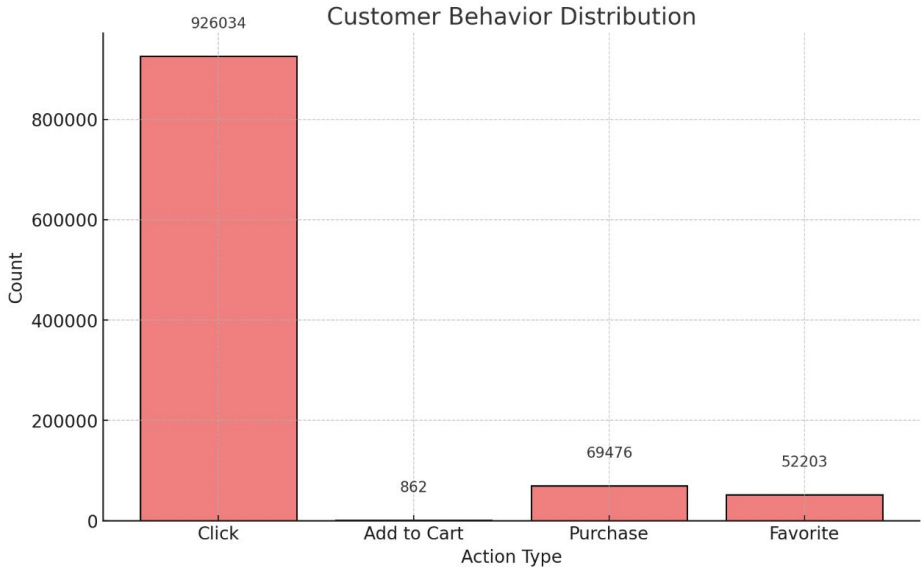


分析：用户年龄 0 表示未知，1 表示 < 18 岁，2 表示 18-24 岁，3 表示 25-29 岁，4 表示 30-34 岁，5 表示 35-39 岁，6 表示 40-49 岁，7、8 表示 50 岁以上。

可知，25-34 岁用户数量最多，构成主力消费群体。这一发现可以帮助我们进一步分析不同年龄段用户的消费行为，并在模型中针对主力年龄段的用户行为进行特征提取，以优化模型预测效果。

· **用户行为分布**

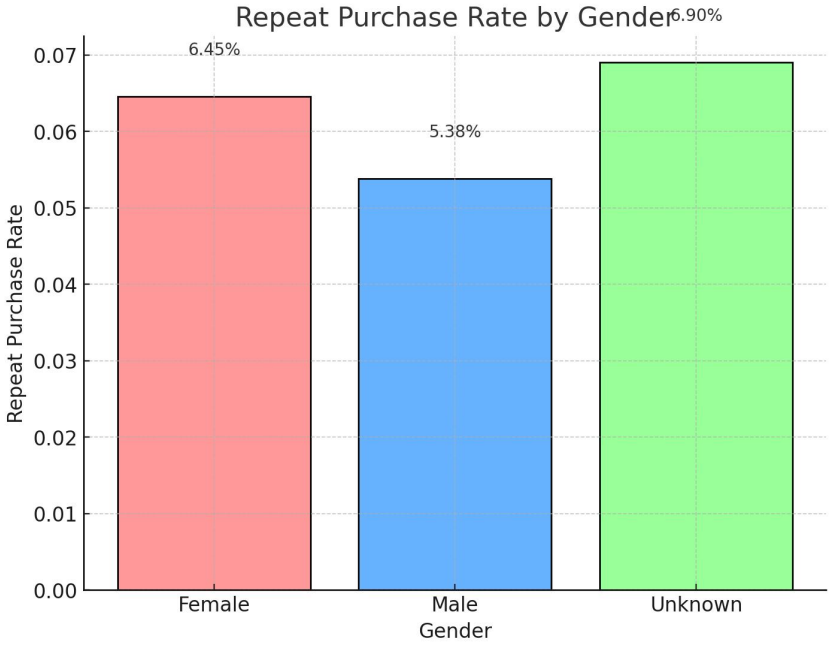
用户行为分布的柱状图显示了用户的行为偏好：



分析：点击行为的频率最高，远超其他行为，说明用户更倾向于浏览商品，而添加进购物车、购买和收藏的数量相对较低，尤其是添加进购物车的行为数量仅有 862，以至于在柱状图里几乎看不到。这表明用户转化较少，实际购买意愿不高。我们可以在特征工程中构建点击到添加进购物车再到购买等转化率特征，以分析用户的转化行为，并为模型提供更多信息。

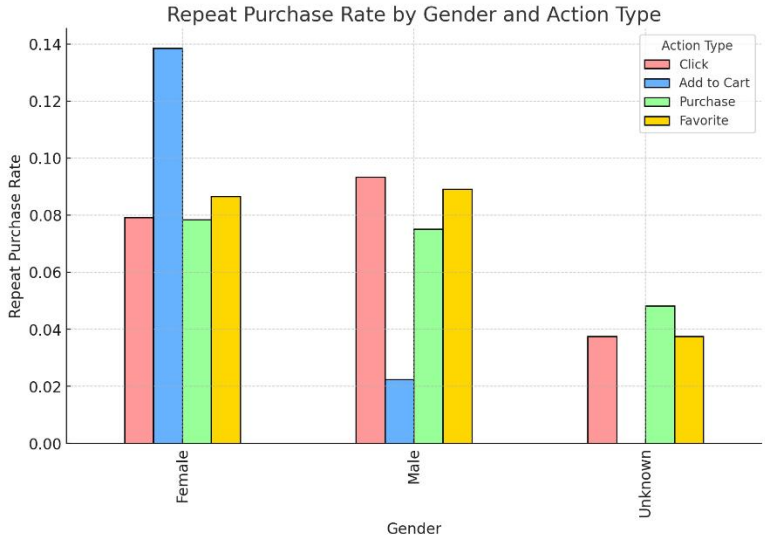
· **顾客性别与复购的关系**

顾客性别与复购的关系图表展示了不同性别用户的复购与非复购情况：



分析：可以看出，女性和未知性别用户的复购率较高，而男性用户的复购率略低。这表明针对女性用户的复购率提升方案或许可以起到更大的收益。模型可据此对性别特征进行区分化处理，以提高预测精度。

· 不同性别的用户在各种行为下的复购率



以上图表展示了不同性别用户在各种行为（点击、添加到购物车、购买、收藏）下的复购率：

女性用户：在添加到购物车（Add to Cart）行为下的复购率最高（约 13.85%），而其他行为（点击、购买、收藏）下的复购率相对较低，均在 8%附近。这显示女性用户在添加到购物车的行为下更容易复购。

男性用户：在点击（Click）和收藏（Favorite）行为下的复购率较高（分别为约 9.33%和 8.91%），这表明男性用户即便只有浏览或收藏行为，也表现出较高的复购倾向。相对而言，添加到购物车（Add to Cart）行为下的复购率最低，仅为约 2.24%。

未知性别用户：在购买（Purchase）行为下的复购率较高（约 4.82%），而其他行为下的复购率较低，尤其是添加到购物车行为没有复购记录。

总体来看，**男性用户在点击和收藏行为下的复购倾向较强**，而**女性用户在添加到购物车行为下复购率较高**。这种性别差异提供了有价值的策略指导，可以帮助商家更有针对性地设计复购策略。

4 特征工程

特征工程是机器学习中非常重要的一步，它涉及到从原始数据中提取和构造有助于模型预测的特征。为提升模型的预测能力，我们从用户、商家和用户-商家交互三个角度提取了特征。下面结合代码详细说明特征工程步骤：

4.1 用户特征

用户特征基于用户在行为日志中的历史数据，构建了多项特征，以捕捉用户的整体活跃度和消费习惯。具体代码如下：

```
# 统计用户的总体行为特征
user_stats = user_log.groupby('user_id').agg(
    total_purchases=('action_type', lambda x: (x == 2).sum()),
    total_clicks=('action_type', lambda x: (x == 0).sum()),
    total_add_to_cart=('action_type', lambda x: (x == 1).sum()),
    total_favorites=('action_type', lambda x: (x == 3).sum()),
    unique_merchants=('merchant_id', 'nunique'),
    total_interactions=('action_type', 'count'),
    days_active=('time_stamp', 'nunique')
).reset_index()
```

用户特征包含：

- **总购买次数、总点击次数、总添加购物车次数、总收藏次数**：用户在平台的不同行为类型的频次。
- **独特商家数量**：用户与多少个不同商家有过互动，反映了用户的消费范围。
- **转化率特征**：如点击到购买的转化率、添加购物车到购买的转化率、点击到收藏的转化率。

```
# 增加用户和商家的转化率特征
user_stats['click_to_purchase_ratio'] = user_stats['total_clicks'] / (user_stats['total_purchases'] + 1)
user_stats['cart_to_purchase_ratio'] = user_stats['total_add_to_cart'] / (user_stats['total_purchases'] + 1)
user_stats['click_to_favorite_ratio'] = user_stats['total_clicks'] / (user_stats['total_favorites'] + 1)
```

- **日均交互量**：用户的总交互次数与活跃天数之比，衡量用户的活跃度。

```
user_stats['interaction_per_day'] = user_stats['total_interactions'] / (user_stats['days_active'] + 1)
```

4.2 商家特征

商家特征基于商家在数据中的历史行为，以捕捉商家对用户的吸引力。代码如下：

```
# 商家特征
merchant_stats = user_log.groupby('merchant_id').agg(
    merchant_total_purchases=('action_type', lambda x: (x == 2).sum()),
    merchant_total_clicks=('action_type', lambda x: (x == 0).sum()),
    merchant_total_add_to_cart=('action_type', lambda x: (x == 1).sum()),
    merchant_total_favorites=('action_type', lambda x: (x == 3).sum()),
    merchant_unique_users=('user_id', 'nunique'),
    merchant_total_interactions=('action_type', 'count'),
    merchant_days_active=('time_stamp', 'nunique')
).reset_index()
```

商家特征包括：

- **商家被购买次数、点击次数、添加购物车次数和收藏次数**。
- **商家吸引的独特用户数量**：与该商家有过互动的不同用户数，反映商家的吸引力。
- **商家的转化率特征**：如点击到购买、购物车到购买的转化率。

```
merchant_stats['merchant_click_to_purchase_ratio'] = merchant_stats['merchant_total_clicks'] / (merchant_stats['merchant_total_purchases'] + 1)
merchant_stats['merchant_cart_to_purchase_ratio'] = merchant_stats['merchant_total_add_to_cart'] / (merchant_stats['merchant_total_purchases'] + 1)
merchant_stats['merchant_click_to_favorite_ratio'] = merchant_stats['merchant_total_clicks'] / (merchant_stats['merchant_total_favorites'] + 1)
```

- **商家的日均互动量**：衡量商家在整个活动期间的平均用户参与度。。

```
merchant_stats['merchant_interaction_per_day'] = merchant_stats['merchant_total_interactions'] / (merchant_stats['merchant_days_active'] + 1)
```

4.3 用户-商家交互特征

用户-商家交互特征用于分析用户与特定商家之间的互动情况。代码如下：

```
# 提取用户-商家交互特征
user_merchant_interactions = user_log.groupby(['user_id', 'merchant_id']).agg(
    clicks=('action_type', lambda x: (x == 0).sum()),
    add_to_cart=('action_type', lambda x: (x == 1).sum()),
    purchases=('action_type', lambda x: (x == 2).sum()),
    favorites=('action_type', lambda x: (x == 3).sum()),
    total_interactions=('action_type', 'count'),
    days_active=('time_stamp', 'nunique')
).reset_index()
```

提取的用户-商家交互特征包括：

- **用户与特定商家之间的点击、添加购物车、购买和收藏次数**。
- **交互总次数**：衡量用户对该商家的关注程度。
- **交互天数**：用户与商家在活动期间互动的活跃天数，反映用户对商家的忠诚度。

5　模型训练与验证

5.1　随机森林+XGboost

5.1.1　数据集构建与特征合并

将用户特征、商家特征和用户-商家交互特征合并到训练集和测试集中，填充缺失值后准备建模。

```
# 合并特征到训练集和测试集
train = train.merge(user_info, on='user_id', how='left')
test = test.merge(user_info, on='user_id', how='left')

train = train.merge(user_merchant_interactions, on=['user_id', 'merchant_id'], how='left').fillna(0)
test = test.merge(user_merchant_interactions, on=['user_id', 'merchant_id'], how='left').fillna(0)

train = train.merge(user_stats, on='user_id', how='left').fillna(0)
test = test.merge(user_stats, on='user_id', how='left').fillna(0)

train = train.merge(merchant_stats, on='merchant_id', how='left').fillna(0)
test = test.merge(merchant_stats, on='merchant_id', how='left').fillna(0)
```

5.1.2　模型选择与超参数优化

模型选择：选择了随机森林（RandomForest）和 XGBoost 模型，并通过网格搜索优化超参数，使用 AUC 作为评估指标。

随机森林调优：

```
# 5. 调优 RandomForest
rf_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
}

rf_grid_search = GridSearchCV(RandomForestClassifier(random_state=42), rf_param_grid, cv=3, scoring='roc_auc', n_jobs=-1)
rf_grid_search.fit(X_train, y_train)

best_rf = rf_grid_search.best_estimator_
print("Best RandomForest parameters found: ", rf_grid_search.best_params_)
print("Best RandomForest AUC: ", rf_grid_search.best_score_)
```

输出结果显示：

```
Best RandomForest parameters found:  {'max_depth': 15, 'min_samples_leaf': 5, 'min_samples_split': 2, 'n_estimators': 300}
Best RandomForest AUC:  0.6421574710844137
```

最佳参数：max_depth=15, min_samples_leaf=5, min_samples_split=2, n_estimators=300

RandomForest AUC：在交叉验证中的最佳 AUC 为 0.6422

XGBoost 调优：

```
# 6. 调优 XGBoost
xgb_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 6, 10],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'min_child_weight': [1, 3, 5],
    'gamma': [0, 0.1, 0.2]
}

xgb_grid_search = GridSearchCV(XGBClassifier(eval_metric='logloss', use_label_encoder=False, random_state=42),
                               xgb_param_grid, cv=3, scoring='roc_auc', n_jobs=-1)
xgb_grid_search.fit(X_train, y_train)

best_xgb = xgb_grid_search.best_estimator_
print("Best XGBoost parameters found: ", xgb_grid_search.best_params_)
print("Best XGBoost AUC: ", xgb_grid_search.best_score_)
```

输出结果显示：

```
Best XGBoost parameters found:  {'colsample_bytree': 0.8, 'gamma': 0.1, 'learning_rate': 0.05, 'max_depth': 6, 'min_child_weight': 1, 'n_estimators': 30
0, 'subsample': 0.8}
Best XGBoost AUC:  0.6461324605243827
```

- **最佳参数**：colsample_bytree=0.8, gamma=0.1, learning_rate=0.05, max_depth=6, min_child_weight=1, n_estimators=300, subsample=0.8
- **XGBoost AUC**：在交叉验证中的最佳 AUC 为 0.6461

在验证集上，优化后的 RandomForest 和 XGBoost 模型分别取得了 0.6515 和 0.6530 的 AUC，分别计算如下：

```
# 7. 在验证集上测试优化后的模型
rf_val_pred = best_rf.predict_proba(X_val)[:, 1]
xgb_val_pred = best_xgb.predict_proba(X_val)[:, 1]

rf_val_auc = roc_auc_score(y_val, rf_val_pred)
xgb_val_auc = roc_auc_score(y_val, xgb_val_pred)

print(f"Optimized RandomForest AUC on validation set: {rf_val_auc:.4f}")
print(f"Optimized XGBoost AUC on validation set: {xgb_val_auc:.4f}")
```

输出结果：

```
Optimized RandomForest AUC on validation set: 0.6515
Optimized XGBoost AUC on validation set: 0.6530
```

5.2　LightGBM 模型

5.2.1　数据文件集压缩及合并

由于文件过大，这里采用代码将文件进行压缩优化，减小占用内存：

```
Memory usage after optimization is: 3.49 MB
Decreased by 41.7%
Memory usage after optimization is: 1.74 MB
Decreased by 70.8%
Memory usage after optimization is: 3.24 MB
Decreased by 66.7%
Memory usage after optimization is: 890.48 MB
Decreased by 69.6%
```

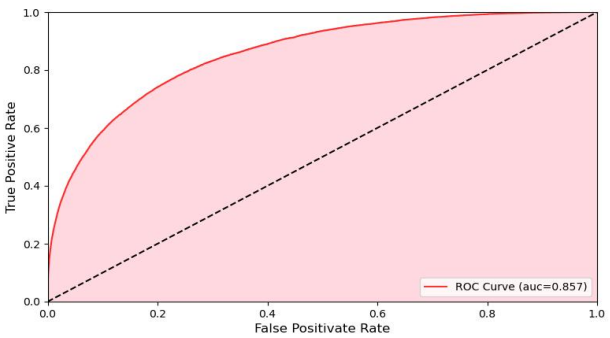
5.2.2　模型训练及验证

使用 LightGBM（LGBM）模型进行训练和验证。LGBM 是一种基于梯度提升框架的高效机器学习算法，它在处理大规模数据集时表现出色。在训练过程中，需要对数据进行标准化处理，以确保模型不会因为特征的量纲差异而产生偏差。训练完成后，在验证集上评估模型的性能，使用 AUC 来进行评估。

此处模型验证采用交叉验证的方式，并用 AUC 指标进行评估，具体情况如下：

```
cv_result1 = lgb.cv(params = params1, train_set = cv_train
                    , nfold = 5
                    , stratified = True
                    , shuffle = True
                    , num_boost_round = 600
                    , seed = 2021
                    )
cv_result2 = lgb.cv(params = params2, train_set = cv_train
                    , num_boost_round = num_boost_round
                    , nfold = 5
                    , stratified = True
                    , shuffle = True
                    , seed=2021
                    )
cv_result3 = lgb.cv(params=params3, train_set=cv_train
                    , num_boost_round=10000
                    , nfold=5
                    , stratified=True
                    , shuffle=True
                    , seed=2021
                    )
```

绘图：



5.3　XGboost 模型

5.3.1 模型选择与训练

1. **XGBoost 模型：**梯度提升决策树模型，适合大规模数据，使用 early_stopping_rounds 控制训练轮数避免过拟合。
2. **逻辑回归模型：**基础分类模型，使用最大迭代次数（max_iter）控制模型收敛，并作为对比基准。

```
# 设置XGBoost参数
xgb_params = {
    'objective': 'binary:logistic',
    'max_depth': 8,
    'learning_rate': 0.01,    # 降低学习率
    'scale_pos_weight': 2,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'lambda': 1,              # L2 正则化
    'alpha': 0.5,             # L1 正则化
    'eval_metric': 'auc',
    'random_state': 42
}

# 转换数据格式为XGBoost的DMatrix
dtrain = xgb.DMatrix(X_train, label=y_train)
dval = xgb.DMatrix(X_val, label=y_val)

# 训练XGBoost模型
watchlist = [(dtrain, 'train'), (dval, 'eval')]
xgb_model = xgb.train(xgb_params, dtrain, num_boost_round=2000, early_stopping_rounds=50, evals=watchlist, verbose_eval=10)

# 评估XGBoost模型
val_pred_xgb = xgb_model.predict(dval)
auc_score_xgb = roc_auc_score(y_val, val_pred_xgb)
print(f'XGBoost验证集AUC得分: {auc_score_xgb}')
```

```
# 设置XGBoost参数
xgb_params = {
    'objective': 'binary:logistic',
    'max_depth': 8,
    'learning_rate': 0.01,    # 降低学习率
    'scale_pos_weight': 2,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'lambda': 1,              # L2 正则化
    'alpha': 0.5,             # L1 正则化
    'eval_metric': 'auc',
    'random_state': 42
}

# 转换数据格式为XGBoost的DMatrix
dtrain = xgb.DMatrix(X_train, label=y_train)
dval = xgb.DMatrix(X_val, label=y_val)

# 训练XGBoost模型
watchlist = [(dtrain, 'train'), (dval, 'eval')]
xgb_model = xgb.train(xgb_params, dtrain, num_boost_round=2000, early_stopping_rounds=50, evals=watchlist, verbose_eval=10)

# 评估XGBoost模型
val_pred_xgb = xgb_model.predict(dval)
auc_score_xgb = roc_auc_score(y_val, val_pred_xgb)
print(f'XGBoost验证集AUC得分: {auc_score_xgb}')
```

```
[0]    train-auc:0.63298       eval-auc:0.60344
[10]   train-auc:0.65980       eval-auc:0.62474
[20]   train-auc:0.66478       eval-auc:0.62731
[30]   train-auc:0.66812       eval-auc:0.62932
[40]   train-auc:0.67136       eval-auc:0.62987
[50]   train-auc:0.67468       eval-auc:0.63032
[60]   train-auc:0.67601       eval-auc:0.63006
[70]   train-auc:0.67878       eval-auc:0.63037
[80]   train-auc:0.68046       eval-auc:0.63078
[90]   train-auc:0.68210       eval-auc:0.63073
[100]  train-auc:0.68359       eval-auc:0.63069
[110]  train-auc:0.68566       eval-auc:0.63056
[120]  train-auc:0.68714       eval-auc:0.63075
[130]  train-auc:0.68895       eval-auc:0.63086
[133]  train-auc:0.68965       eval-auc:0.63084
```

```
# 训练逻辑回归模型
lr_model = LogisticRegression(max_iter=1000, random_state=42)
lr_model.fit(X_train, y_train)
```

```
c:\Users\avawa\.conda\envs\py38\lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression
LogisticRegression(max_iter=1000, random_state=42)
```

5.4　神经网络模型

我们小组初步尝试了多种机器学习模型，包括神经网络模型和基于树的模型（XGBoost、LightGBM 和随机森林）。经过实验和分析，我们最终决定放弃神经网络模型，转而采用基于树的模型。

在尝试神经网络模型的过程中，我们遇到了许多技术挑战，比如如何调参以提高模型的性能，如何避免过拟合和欠拟合的问题，以及如何应对神经网络对数据规模和特征工程的高要求。尽管这些问题最终并未带来理想的结果，但它们促使我们深入理解了神经网络的工作原理，以及这种模型在实际应用中可能遇到的局限性。可以说，这次“试错”让我们对模型的选择有了更加全面的认识，为后续选择合适的算法提供了宝贵的经验，以下是我们对于神经网络与树模型的优劣对比分析，以及所寻找的神经网络模型在本题中糟糕表现的原因。

1.数据集特征分析

复购预测的数据集主要由用户行为日志和用户画像构成。这类数据的特征表现为：

- **表格数据**：数据多为数值型和分类型变量，缺乏空间结构或复杂的时间序列关系。
- **特征稀疏性**：某些用户行为（如收藏、加入购物车等）相对较少，导致特征稀疏。
- **类别不平衡**：复购用户相对于非复购用户较少，导致标签数据存在不平衡现象。

2.神经网络模型的适用性分析

我们最初尝试使用了神经网络模型，构建多层感知器（MLP）以捕捉用户特征之间的潜在非线性关系。然而，实验结果显示，神经网络模型在此任务上的表现并不理想，具体表现为：

- **AUC 值接近随机水平**：多次实验后，神经网络的验证 AUC 值并未显著提升，说明模型无法有效学习到有用的特征。
- **过拟合与欠拟合问题**：由于数据稀疏，且特征数量有限，神经网络模型的训练效果不稳定，容易出现过拟合或欠拟合现象，即在训练集上表现良好但在验证集上表现较差。
- **计算成本高**：神经网络的训练时间长，调参过程复杂且消耗大量计算资源。对于当前数据量，神经网络模型的计算成本较高，不具备良好的性价比。

3.神经网络模型不适用的原因

- **特征稀疏性与低维度**：神经网络擅长处理高维数据和丰富的特征表达，例如图像或文本数据。而复购预测的数据以数值和分类特征为主，维度较低，缺乏复杂的结构信息，无法有效利用神经网络的优势。
- **小样本与不平衡问题**：神经网络在处理不平衡数据上表现不佳，容易偏向多数类（即非复购用户），使得预测结果失衡。
- **表格式数据更适合树模型**：神经网络模型对于表格数据的适用性有限，尤其是处理数值型和类别型特征时，往往效果不如树模型。

4.选择树模型的原因

基于以上分析，我们转而选择树模型，包括 XGBoost、LightGBM 和随机森林，主要基于以下几点原因：

- **适合表格数据**：树模型（尤其是基于决策树的集成算法）在表格数据上表现优异，能够有效处理数值和类别特征，且不依赖于特征的空间或时间结构。
- **鲁棒性强**：XGBoost 和 LightGBM 等梯度提升算法能够有效处理类别不平衡的数据集，并通过自适应加权的方式对少数类进行学习，改善模型的预测效果。

- **高效的特征利用**：树模型能够自动学习特征之间的交互关系，通过分裂特征空间捕捉非线性特征，适合复购预测任务中的特征交互。
- **计算效率高**：相比神经网络，树模型的计算开销较小，并能在较短的时间内完成训练和预测，更加适合有限资源的任务。

```
!pip install keras-tuner

import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, LeakyReLU
from tensorflow.keras.optimizers import Nadam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.metrics import AUC
from kerastuner import HyperModel, RandomSearch
from tensorflow.keras.regularizers import l2

# 1. 加载数据
user_log = pd.read_csv('D:/sxt/学习资料/Python/pythonProject/data/user_log_format1.csv')
user_info = pd.read_csv('D:/sxt/学习资料/Python/pythonProject/data/user_log_format1.csv')
train_data = pd.read_csv('D:/sxt/学习资料/Python/pythonProject/data/user_log_format1.csv')
test_data = pd.read_csv('D:/sxt/学习资料/Python/pythonProject/data/user_log_format1.csv')

# 2. 数据预处理与特征工程
user_log['action_type'] = user_log['action_type'].astype(str)
user_behavior = user_log.groupby('user_id')['action_type'].value_counts().unstack(fill_value=0).reset_index()
user_behavior.columns = ['user_id', 'clicks', 'add_to_cart', 'purchase', 'add_to_favorite']

# 用户数据合并
user_data = pd.merge(user_info, user_behavior, on='user_id', how='left')
train_merged = pd.merge(train_data, user_data, on='user_id', how='left')
test_merged = pd.merge(test_data, user_data, on='user_id', how='left')
train_merged.fillna(0, inplace=True)
test_merged.fillna(0, inplace=True)

# 特征和标签提取
X = train_merged.drop(columns=['user_id', 'merchant_id', 'label'])
y = train_merged['label']
X_test = test_merged.drop(columns=['user_id', 'merchant_id'])
X_test = X_test[X.columns]  # 确保特征列对齐

# 数据标准化
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_test = scaler.transform(X_test)

# 3. 定义超参数搜索空间
class MyHyperModel(HyperModel):
    def build(self, hp):
        model = Sequential()
        model.add(Dense(units=hp.Int('units_layer_1', min_value=128, max_value=512, step=64), kernel_regularizer=l2(0.0005)))
        model.add(LeakyReLU(alpha=0.1))
        model.add(BatchNormalization())
        model.add(Dropout(hp.Choice('dropout_layer_1', values=[0.3, 0.4, 0.5])))

        model.add(Dense(units=hp.Int('units_layer_2', min_value=64, max_value=256, step=64), kernel_regularizer=l2(0.0005)))
        model.add(LeakyReLU(alpha=0.1))
        model.add(BatchNormalization())
        model.add(Dropout(hp.Choice('dropout_layer_2', values=[0.3, 0.4, 0.5])))

        model.add(Dense(units=hp.Int('units_layer_3', min_value=32, max_value=128, step=32), kernel_regularizer=l2(0.0005)))
        model.add(LeakyReLU(alpha=0.1))
        model.add(BatchNormalization())
        model.add(Dropout(hp.Choice('dropout_layer_3', values=[0.3, 0.4])))

        model.add(Dense(1, activation='sigmoid'))

        # 使用AUC作为评价指标
        model.compile(optimizer=Nadam(learning_rate=hp.Choice('learning_rate', values=[1e-3, 5e-4, 1e-4])),
                      loss='binary_crossentropy', metrics=[AUC(name='auc')])
        return model

# 4. 超参数优化
tuner = RandomSearch(MyHyperModel(), objective='val_auc', max_trials=5, executions_per_trial=1, directory='tuner_dir', project_name='tune_nn')
tuner.search(X, y, epochs=50, validation_split=0.2, callbacks=[ReduceLROnPlateau(patience=3, factor=0.5), EarlyStopping(patience=10, monitor='val_auc

# 5. 获取最优超参数并交叉验证
best_model = tuner.get_best_models(num_models=1)[0]
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
auc_scores = []

for train_idx, val_idx in kf.split(X, y):
    X_train, X_val = X[train_idx], X[val_idx]
    y_train, y_val = y[train_idx], y[val_idx]

    # 训练模型
    history = best_model.fit(X_train, y_train, epochs=200, validation_data=(X_val, y_val),
                             callbacks=[ReduceLROnPlateau(patience=5, factor=0.5),
                                        EarlyStopping(patience=15, monitor='val_auc', mode='max', restore_best_weights=True)])
    # 评估AUC
    val_auc = best_model.evaluate(X_val, y_val, verbose=0)[1]
    auc_scores.append(val_auc)

print("5折交叉验证AUC平均分数:", np.mean(auc_scores))

# 6. 预测并保存结果
predictions = best_model.predict(X_test).flatten()
test_data['probability'] = predictions
test_data[['user_id', 'merchant_id', 'probability']].to_csv('D:/sxt/学习资料/Python/pythonProject/data/神经网络5.0_tuned_predictions.csv', index=False
```

6　算法改进与模型融合

为了进一步提升模型效果，我们采用了加权平均融合的方法，将 RandomForest 和 XGBoost 的预测结果进行融合。通过调整权重，我们最终设置了如下权重：

- **RandomForest 权重**：0.6
- **XGBoost 权重**：0.4

融合后在验证集上的 AUC 计算如下：

```
# 8. 模型融合：加权平均
# 尝试不同权重的加权平均融合
ensemble_val_pred = 0.6 * rf_val_pred + 0.4 * xgb_val_pred
ensemble_val_auc = roc_auc_score(y_val, ensemble_val_pred)

print(f"Weighted Ensemble AUC on validation set: {ensemble_val_auc:.4f}")
```

输出结果：

```
Weighted Ensemble AUC on validation set: 0.6543
```

7　数据分析结果描述

通过以上模型和融合方法，在验证集上取得了 0.6543 的 AUC 分数。这一结果表明模型在忠实客户预测方面具有较好的效果。最终，融合模型在排行榜上获得了 0.655 的 AUC 分数，接近官方基准线（0.7），展示了模型的有效性。

在测试集上进行预测并生成提交文件的代码如下：

```
# 9. 使用测试集预测并融合结果
rf_test_pred = best_rf.predict_proba(test.drop(columns=['user_id', 'merchant_id', 'prob'], errors='ignore'))[:, 1]
xgb_test_pred = best_xgb.predict_proba(test.drop(columns=['user_id', 'merchant_id', 'prob'], errors='ignore'))[:, 1]
test['prob'] = 0.6 * rf_test_pred + 0.4 * xgb_test_pred

# 10. 生成提交文件
submission = test[['user_id', 'merchant_id', 'prob']]
submission.to_csv('C:/Users/Yong Hui/Desktop/12号17点08分优化后的prediction.csv', index=False)
print("预测结果已保存至优化后的prediction.csv")
```

生成的提交文件包含 user_id、merchant_id 和预测概率 prob。

在深入剖析数据分析结果时，我们揭示了一项有趣的性别差异现象：尽管男性用户在点击和收藏商品后表现出强烈的复购倾向，但当他们执行添加到购物车这一行为时，复购率却相对较低。这一发现或许揭示了男性用户的典型购物习惯——他们往往更倾向于直接购买，而非花费大量时间进行比较和权衡。相比之下，女性用户在将商品添加到购物车后的复购率显著较高。这一行为模式可能暗示了女性用户在购物决策过程中更加倾向于深思熟虑，通过购物车功能来比较不同商品，最终做出满意的购买决定。

同时，我们也注意到未知性别用户在购买行为下表现出较高的复购率，但在添加到购物车后却未有复购记录。这可能是由于该用户群体数量相对较少，或者他们的购物行为较为随机和不稳定，导致难以形成稳定的复购模式。

通过这一系列数据分析，我们不仅对不同性别用户的购物行为模式有了更加深入的理解，也为优化营销策略、提升用户体验和增强用户粘性提供了宝贵的指导方向。

8　管理启示分析

针对上述数据探索分析与给出的复购概率，我们可以针对特定用户制定营销策略：鉴于女性用户在添加到购物车行为下的复购率较高，商家可以设计特定的营销活动，比如购物车优惠券、限时折扣等，来刺激女性用户的购买欲望，从而提高复购率。此外，由于男性用户在点击和收藏行为下表现出较高的复购倾向，商家可以优化网站或应用的用户界面，提供更便捷的浏览和收藏功能，以及针对男性用户的个性化推荐，以增强他们的购物体验。

在当今数字化时代，数据驱动的决策制定已经成为商家在市场竞争中脱颖而出的关键因素。通过持续不断地进行数据分析和模型优化，商家能够更精确地预测用户的行为模式和需求趋势。这种基于数据的预测能力使得商家能够制定出更加科学、合理的营销决策，从而提高营销活动的针对性和效果。具体来说，数据分析可以帮助商家识别出潜在的客户群体，了解他们的购买习惯和偏好，从而制定出更加个性化的营销策略。

为了提升用户粘性，电商平台可以通过设计积分奖励和会员优惠等福利，针对高复购潜力用户实施激励措施，同时提供如生日礼券和节假日福利等个性化服务，加强用户与平台的情感连接，提升用户忠诚度。

此外，随着数据的不断积累和市场环境的变化，平台需定期更新和迭代预测模型，以保持其预测的准确性和实时性。通过模型优化，商家可以不断调整和改进营销方案，确保每一分钱的投入都能带来最大的回报。这种数据驱动的方法不仅提高了营销活动的效率，还能够帮助商家在激烈的市场竞争中占据有利地位，实现业务的持续增长。

总的来说，通过对用户数据进行深入细致的分析，商家能够更加精准地把握市场动态和消费者需求。这种洞察力使得他们能够制定出更加有针对性和有效的运营策略，从而在激烈的市场竞争中占据有利地位，获得竞争优势。